COMAL 80 — Quick Reference

Target: an experienced programmer learning COMAL 80 on the Commodore 64.

Summary - COMAL is an interactive, structured language mixing BASIC-style immediacy with Pascal-like structured constructs. Use `USE <package>` to enable device/graphics/sound extensions and `DISCARD` to remove them.

Workflow & editing - Interactive orders: `PRINT`, `INPUT`, `ZONE`, `PAGE`. - Program cycle: edit (enter lines) → `SCAN` (check/define procedures) → `RUN`. - Handy system commands: `NEW`, `AUTO [start,step]`, `RENUM`, `LIST`, `SAVE "name"`, `LOAD "name"`, `VERIFY`, `SIZE`, `CAT`/`DIR`.

Syntax basics - Comments: `//` at start of a line or after a statement. - Assignment: `name := expression`. - Identifiers: numeric names (default) and string names ending in `$` (e.g. `s$`). Names may be long; use `'` (apostrophe) to bind multiword procedure names (e.g. `PROC start'graphics`). - Arrays and strings: `DIM arr OF n` (array or string), `DIM s$ OF 25` reserves string space.

Primitive types & conventions - Numeric (floating) and string (`$`). - Boolean constants: `TRUE` (1), `FALSE` (0).

Operators - Arithmetic: `+ - * /` and `EXP` (exponent), `SQR()` available as function. - Integer operations: `DIV` (integer divide), `MOD` (remainder). - Relational: `= <> < > <= >=`. - Logical: `NOT`, `AND`, `AND THEN`, `OR`, `OR ELSE`, `IN`. - Bitwise: `BITAND`, `BITOR`, `BITXOR`.

Built-in functions (common) - Numeric: `INT(x)`, `SGN(x)`, `SQR(x)`, `PI`, `COS(x)`, `SIN(x)`, `TAN(x)`, `ATN(x)`, `RND(a,b)` (random in range), `RANDOMIZE`.

Contents

- Common (base language)
- Turtle package (`USE turtle`)
- Graphics package (`USE graphics`)
- File / Disk / Datassette (I/O)
- System & low-level
- Sound / Sprites / Font / Peripheral packages (overview)

Common (base language)

USE: (none — core language)

Overview - Core control structures, program management, basic I/O, strings, arrays, math and system calls.

Program & editing

Syntax

```
NEW
AUTO [start,step]
RENUM
LIST
SCAN
RUN
SAVE "name"
LOAD "name"
VERIFY
SIZE
CAT | DIR
```

Example

```
AUTO 100,10
// write program lines
SCAN  // verify structure
RUN
```

Notes - `AUTO` with parameters sets starting line and increment. `CAT`/`DIR` list disk contents.

Control flow

Syntax & examples

```
IF <cond> THEN
  ...
[ELIF <cond> THEN ...]
[ELSE ...]
ENDIF

CASE <expr> OF
  WHEN val1
    ...
  WHEN val2
    ...
  [OTHERWISE ...]
ENDCASE

FOR i := start TO end [STEP s] DO
  ...
ENDFOR i

WHILE <cond> DO
  ...
ENDWHILE

REPEAT
  ...
UNTIL <cond>
```

Example

```
FOR i := 1 TO 4 DO
  FORWARD(75)
  RIGHT(90)
ENDFOR i
```

Procedures & functions

Syntax

```
PROC name(params)
  ...
ENDPROC name

FUNC name(params)
  ...
  RETURN expr
ENDFUNC name
```

Example

```
PROC polygon(sides,len)
  FOR i := 1 TO sides DO
    FORWARD(len)
    RIGHT(360/sides)
  ENDFOR i
ENDPROC polygon
```

I/O and formatting

Syntax

```
PRINT expr [, expr2 ; expr3 ...]
ZONE n
PAGE
INPUT "prompt": var
PRINT AT row,col expr
INPUT AT row,col,width "prompt": var
READ / DATA / RESTORE
```

Notes - `,` (comma) uses `ZONE` spacing; `;` prints a single space. `READ` consumes `DATA` records sequentially.

Expanded I/O details (parameters, behavior)

PRINT - Syntax: `PRINT item1 [, item2 ; item3 ...]` - Parameters: - `item`: string literal, numeric expression, string variable (ending `$`), or function call. - Multiple items separated by `,` are placed into columns controlled by `ZONE`. Items separated by `;` are printed with a single space. - Notes & side-effects: - Trailing semicolon prevents newline; omit to advance to next line. - `PRINT AT r,c expr` places output at row `r`, column `c` (1-based). Valid ranges depend on text mode (commonly 1..25 rows, 1..40/80 columns). - Example:

```
ZONE 10
PRINT "Value:", x, "units"
```

```
PRINT AT 5,10 "Status:"; status$
```

INPUT - Syntax: `INPUT "prompt": var` or `INPUT AT r,c,w "prompt": var` - Parameters: - `prompt`: string shown before input. - `var`: target variable (numeric or string). For strings, use `var$` and ensure `DIM var$ OF n` reserved enough space. - `r,c,w`: when using `INPUT AT`, `r` and `c` are row and column (1-based); `w` is width (number of characters allowed) for string input. - Behavior: - Pauses program until user types a value and presses . - For multiple comma-separated input prompts, accept comma-separated responses. - Numeric input into non-numeric variables may cause conversion or error depending on implementation. - Examples:

```
INPUT "Enter number": n
INPUT AT 10,5,3 "Ch:" : ch$
```

ZONE - Syntax: `ZONE n` - Parameter: `n` numeric column width used when `PRINT` uses commas to separate fields. - Example: `ZONE 12` sets each comma-separated field to 12-character width.

READ / DATA / RESTORE - Syntax: `DATA item1,item2,... ; READ var1,var2,... ; RESTORE [label]` - Parameters & behavior: - `DATA` lines embed constants in program. - `READ` sequentially consumes values from `DATA` until exhausted. - `RESTORE` resets the read pointer to the first `DATA` or to `DATA` with `label`. - Example:

```
DATA 10,20,30
READ a,b,c
RESTORE
```

SELECT INPUT / SELECT OUTPUT - Syntax: `SELECT INPUT "device" ; SELECT OUTPUT "device"` - Parameters: - `device`: device identifier string (examples: `"0:"`, `"1:"`, `"prn"`, or cartridge-specific device names). Exact device names depend on the platform/cartridge. - Notes: - Redirects subsequent `INPUT`/`PRINT` calls to the named device until changed. - Use `SELECT OUTPUT "0:"` to send output to disk files where the cartridge supports it.

Strings, arrays & basic functions

Syntax / examples

```
DIM s$ OF 25
DIM arr OF 100
LEN(s$) ; VAL(s$) ; STR$(n) ; CHR$(n) ; ORD(ch)
INSTR(sub,str)   // returns index or 0
```

Math & random

Syntax / examples

```
INT(x) ; SGN(x) ; SQR(x) ; PI ; COS(x) ; SIN(x) ; TAN(x) ; ATN(x)
DIV, MOD
RANDOMIZE
RND(a,b)    // COMAL 80 supports integer-range usage
```

System / low-level

Syntax / examples

```
PEEK(addr) ; POKE(addr,val)
SYS addr[,params]
STOP ; END ; NULL
TRAP / HANDLER / ENDTRAP
```

Notes - `PEEK`/`POKE` and `SYS` are platform-specific (C-64 memory map). `NULL` is a no-op useful in tight loops.

Turtle package

USE: `USE turtle`

Overview - Turtle provides pen-based drawing (a drawing cursor / “turtle”) and helpers for split-screen text/graphics workflows. Use `USE turtle` to load the package; `DISCARD` removes it.

Screen & modes

Syntax

```
SPLITSCREEN
GRAPHICSCREEN(n)
```

Parameters & notes - `SPLITSCREEN`: enables a split view showing several lines of editable text at the top and a graphics canvas below. No parameters. - `GRAPHICSCREEN(n)`: select graphics screen mode. Typical values: - `0` — full-text mode (no graphics canvas) - `1` — split text+graphics (or the opposite depending on cartridge) - other values: implementation dependent; consult package docs. Side effect: changes where subsequent graphics commands draw; also affects `PRINT AT` coordinates.

Movement & pen (basic drawing)

Syntax

```
FORWARD(n)  ; FD(n)
BACK(n)     ; BK(n)
LEFT(d)     ; LT(d)
RIGHT(d)    ; RT(d)
SETHEADING(h) ; SETH(h)
HOME
```

Parameters & notes - `n` (distance): numeric. Units are package-defined “steps” or pixels; typical range fits on-screen coordinates (0..~320 horizontally on C64 high-res). Values outside the canvas frequently wrap or clip depending on package behavior. - `d` (degrees): numeric. Angle in degrees; positive values rotate the turtle by that many degrees. Angles may be any numeric value; 360 equals full rotation. - `h` (heading): numeric.

Absolute heading in degrees (commonly 0 = up, 90 = right, 180 = down, 270 = left). Some implementations use 0 = right; check package docs. - `HOME`: centers the turtle at canvas center and (usually) resets heading to default (implementation-dependent). - Movement commands typically draw only when the pen is down; use `PENUP` to move without drawing.

Pen, color, and appearance

Syntax

```
PENUP / PU   // lift pen
PENDOWN / PD // lower pen
PENCOLOR(c)  ; PC(c)
BACKGROUND(c) ; BG(c)
SHOWTURTLE / ST
HIDETURTLE / HT
CLEARSCR / CS
```

Parameters & notes - `PENUP` / `PENDOWN`: no parameters. `PENDOWN` enables drawing during movement; `PENUP` temporarily disables drawing. - `PENCOLOR(c)`: `c` is a numeric color code. Color palette and numeric mapping are in Appendix B; typical range 0..15 on C-64. Setting pen color affects lines and fill operations performed while that pen is active. - `BACKGROUND(c)`: sets the background color for the graphics canvas; side effect may clear or repaint the canvas depending on package. - `SHOWTURTLE` / `HIDETURTLE`: toggles visibility of the turtle cursor. Does not affect drawing. - `CLEARSCR` / `CS`: clears the graphics canvas; often does not modify the text area when in split-screen mode.

Common behavior & gotchas - Coordinates, orientation and screen mapping are package-dependent; many COMAL turtle commands assume a logical drawing space that maps to device pixels through the package. Expect slight differences between COMAL distributions. - Some turtle packages wrap coordinates: moving past an edge may reappear on opposite edge. Others clamp or clip drawing.

Graphics package

USE: `USE graphics`

Overview - `graphics` exposes lower-level drawing primitives: coordinate drawing, fills, circles, pixel plotting and (in some cartridge variants) sprite and bitmap helpers. Load with `USE graphics`.

Screen selection

```
GRAPHICSCREEN(n)
SPLITSCREEN
```

- `GRAPHICSCREEN(n)`: selects which graphics buffer or mode to use.

Common meanings:
  - `0` — text-only or character-graphics
  - `1` — high-resolution bitmap graphics
  - other modes available on extended packages (multicolor or alternate pages)
  - Side effects: changes coordinate origin, resolution and color interpretation for subsequent drawing calls.

Drawing primitives

```
MOVETO(x,y)
DRAWTO(x,y)   // draw line from current point to (x,y)
CIRCLE(x,y,r)
PAINT(x,y)    // flood-fill region containing (x,y)
PLOT(x,y)     // set pixel (implementation dependent)
POINT(x,y)    // read pixel color (if provided)
```

Parameters & notes - `x,y`: numeric coordinates. Origin (0,0) location depends on mode (commonly top-left or center). Valid ranges map to screen resolution (e.g., 0..319 for x on high-res); check package docs. - `r` (radius): numeric. Positive values draw circle; fractional values may be rounded. Circles overlapping screen edges may be clipped. - `PAINT(x,y)` / `#111(x,y)`: flood-fill starting at pixel (x,y) using the current pen color. Note that `PAINT` requires the region to be enclosed; behavior on open regions is package-dependent. - `MOVETO` sets the current drawing cursor without drawing; `DRAWTO` draws a line and updates the cursor. - `PLOT` / `POINT` names vary by implementation — some packages call these `PSET`/`PGET`.

Color & attributes

```
PENCOLOR(c)
BACKGROUND(c)
```

- `c` is a numeric color index; consult Appendix B for the color map. Some graphics modes support fewer colors (e.g., monochrome or multicolor packing).

Sprites & advanced features

- Sprite control is typically provided by a separate sprite package or additional `graphics` orders (e.g., `SPRITE ON`, `SPRITE X,Y`, `SPRITE COLLIDE`). Parameters include sprite number (index), (x,y) position, and attribute flags for multi-color.
- Saving/restoring drawings: some packages provide `SAVE IMAGE` or `WRITE FILE` helpers; otherwise use the disk/file API to save raw bitmap blocks.

Examples

```
USE graphics
GRAPHICSCREEN(1)
MOVETO(100,100)
DRAWTO(160,100)
```

```
CIRCLE(160,100,50)
PENCOLOR(2)
PAINT(160,100)
```

Notes - Exact function names and parameter semantics vary across COMAL cartridge versions. When in doubt, consult the manual appendix for your cartridge and the `?` or `HELP` feature (if present) in the runtime.

File / Disk / Datassette (I/O)

USE: often none for basic `SAVE`/`LOAD`; low-level disk commands are passed with `PASS`.

Program & disk operations

```
SAVE "name"            // program
LOAD "name"
CAT | DIR               // directory listing
PASS "<command>"       // forward command to DOS (e.g., format)
SIZE                    // memory/disk stats (implementation-dependent)
```

Expanded disk/tape and file I/O details

SAVE / LOAD - Syntax: `SAVE "name"` ; `LOAD "name"` - Parameters & notes: - `name` may be `"cs:program"` for cassette or `"program"` for disk depending on device. - On disk, `SAVE` writes to the currently selected drive; use `PASS` or device-specific qualifiers to select drive. - `LOAD` restores a saved program to memory; existing program in memory is overwritten unless `MERGE` or `ENTER` forms are available. - Example:

```
SAVE "polygon"
LOAD "polygon"
```

PASS (DOS / low-level command passthrough) - Syntax: `PASS "<text-command>"` - Purpose: forwards raw text to the underlying DOS or tape handler (useful for formatting, disk commands not exposed by COMAL orders). - Example: `PASS "n0MYDISK,XX"` — common convention to format drive 0 with name `MYDISK` and ID `XX` (implementation-specific).

High-level file I/O (COMAL API) - Note: exact syntax and features vary by COMAL/disk DOS combination. The forms below are canonical descriptions; check the cartridge manual for exact keywords and parameter order.

Common operations

```
CREATE "filename"        // create new sequential file
OPEN "filename"          // open file for read/write/append (mode may be
required)
READ var1,var2,...        // read next record or fields
WRITE var1,var2,...       // write record or fields
APPEND                    // append-mode operations
RANDOM <record>           // open or address random-access records
```

```
CLOSE                      // close file
```

Parameters & examples - `CREATE "data.seq"`: creates a new sequential file named `data.seq`. - `OPEN "data.seq"` may accept optional mode flags (e.g., `FOR INPUT`, `FOR OUTPUT`, `FOR RANDOM`) depending on implementation. - `READ` / `WRITE`: for sequential files, these read or write the next record; for random-access, they operate on a current record pointer.

Examples (conceptual)

```
CREATE "scores"          // create file
OPEN "scores"            // open for writing
WRITE name$, score
CLOSE

OPEN "scores"            // open for reading
READ name$, score
CLOSE
```

Notes (File I/O) - Random-access files require fixed record length and addressing. `RANDOM` statements often need a record size parameter and a record-number index when reading/writing. - File APIs differ between tape and disk; tape formats are usually sequential only.

File I/O (high-level)

```
MOUNT
CREATE "filename"
OPEN "filename"
READ / WRITE / APPEND
RANDOM  // random access
CLOSE
```

Notes - Exact `OPEN`/`RANDOM` syntax depends on DOS/cartridge extensions—see Chapter 6 of the manual for full examples.

System & low-level

USE: (none required for the core low-level calls)

Key orders

```
PEEK(addr) ; POKE(addr,val)
SYS addr[,params]
TRAP <label> ; HANDLER ... ENDTRAP
TRACE (where supported)
```

Detailed system / low-level notes

PEEK / POKE - Syntax: `v := PEEK(addr)` ; `POKE(addr, val)` - Parameters: - `addr`: numeric memory address (0..65535 on 16-bit machines like the C-64). Must be integer. - `val`: numeric byte value to write (0..255). On POKE, values outside range may be truncated. - Example: `POKE 53280,2` sets the C-64

screen border color to color code 2. - Notes & cautions: - Writing to hardware registers or ROM areas can crash the system or corrupt data. - Reading/writing should respect address and value ranges.

SYS - Syntax: `SYS addr[,params]` - Purpose: call a machine-language routine at absolute memory address `addr`. - Parameters & conventions: - Passing parameters and interpreting results is platform-specific (usually via CPU registers or zero-page addresses). COMAL does not standardize calling conventions for `SYS` — consult the machine code module documentation. - `SYS` may not return to the same interpreter state if the ML routine modifies system vectors; use with care and only when you understand the module. - Example: `SYS 49152` (calls machine code at $C000).

TRAP / HANDLER / ENDTRAP - Syntax:

```
TRAP <label-or-number>
...  // code that may cause runtime error
HANDLER
  ... // error handling code
ENDTRAP
```

- Behavior:
  - `TRAP` begins a region where runtime errors are intercepted and control passes to the `HANDLER` when an error occurs.
  - The `label-or-number` may specify an error number or handler label depending on COMAL version.
- Example (conceptual):

```
TRAP 100
  OPEN "nonexistent"
HANDLER
  PRINT "Error during open"
ENDTRAP
```

ERR, ERRFILE, ERRTEXT - These symbols and functions exist to obtain error codes, file-specific error info, and textual messages. Exact usage depends on COMAL variant.

TRACE - If supported, `TRACE` enables execution tracing to help debugging; exact output and controls depend on the COMAL environment.

Notes (System): - Low-level operations are powerful but platform-dependent; document addresses, registers and side-effects when mixing `SYS`/`POKE` with high-level COMAL code.

Notes - `PEEK`/`POKE` and `SYS` are inherently platform-specific; using them requires knowledge of the C-64 memory map and caution to avoid crashes.

Sound / Sprite / Font / Peripheral packages (overview)

USE: `USE sound`, `USE sprites`, `USE font`, etc. — package names depend on

cartridge distribution.

Typical provided orders (package-dependent): - Sound: `SOUND(n,f,d)` / `PLAY` / music pattern routines. - Sprites: `SPRITE ON/OFF`, `SPRITE X,Y`, `SPRITE COLLIDE`, multi-color sprite rules. - Font: `REPLACECHAR`, `LOADFONT`, `SETFONT` (procedures to replace character set). - Peripherals: `JOY`, `PADDLE`, `LIGHTPEN`, `RS232` series of orders.

Examples & usage tips (sound/sprites/peripherals)

- Sound sequences: use high-level `PLAY` patterns where available; `SOUND` is most useful for single tones or quick effects. Typical `SOUND(channel,frequency,duration)` calls expect `channel` (integer voice), `frequency` (Hz or encoded tone) and `duration` (time units). Check package docs for encoding and timing units.
- Sprite layering & motion: typical sprite operations accept an index, x and y coordinates and optional attribute flags. Example pattern (conceptual):

```
USE sprites
SPRITE ON 1
SPRITEX 1,100   // set sprite 1 X position
SPRITEY 1,50    // set sprite 1 Y position
SPRITE COLOR 1,2
```

- Joystick and paddle input: `JOY(port)` usually returns a bitmask of directional/buttons; `PADDLE(port)` returns an analog reading (commonly 0..255). Poll or read in a `WHILE` loop to detect input.
- Lightpen: lightpen APIs vary; calls typically return `x,y` coordinates and may set a trigger flag. Use `INPUT AT`/`PRINT AT` with coordinates to display or accept lightpen input.
- Serial / RS-232: packages supply `OPEN`, `CLOSE`, `READ`, `WRITE` forms for serial ports with configuration parameters (baud, parity, data bits, stop bits). Example (conceptual): `OPEN "RS:1,19200,N,8,1"` — format differs by extension.

Notes (Packages): - These packages vary by COMAL cartridge and region; consult Chapter 5 and the package-specific documentation in the manual for full API and variable lists (e.g., color codes, sprite indices, sound channel mapping).

Notes on limitations and OCR caveats

- This reference is distilled from an OCRed manual and rephrased into intended COMAL 80 syntax. Some package names and detailed parameter lists vary by COMAL cartridge version and regional

releases. For exhaustive, formal definitions consult the original manual appendices or the COMAL Handbook.

Next steps - I can split each library into its own markdown file (e.g., `comal80-turtle.md`) or export this grouped reference as JSON/YAML for tooling. Tell me which format you prefer.